Assignment 4 (Written): Dataflow Analysis

17-654/17-754: Analysis of Software Artifacts
Jonathan Aldrich (`jonathan.aldrich@cs.cmu.edu`)

Due: Monday, February 20, 2006 (5:00 pm)

100 points total

Turn in a file named <username>-17654-A4.{pdf,doc}, where username is your Andrew id, or hand in a paper response in class. At the top of the document, state your name, Andrew id, and how long you spent on the assignment.

Assignment Objectives:

- Precisely define two analyses using lattices, abstraction functions, and flow functions.
- Derive the control flow graph of a program.
- Simulate analysis execution on a program using the worklist algorithm.

1 Lock Analysis (40 points)

After graduating from CMU, you have been hired by FluidSoft, a (fictional) company devoted to bringing the benefits of the Fluid Java analysis tools to C programmers. Your first task is to get a simple lock analysis up and running.

You quickly observe that C presents a harder problem than Java, because there's no built-in synchronization statement. Thus it's easy to make simple errors that can't happen in Java, like locking a lock when you enter a function and forgetting to unlock it when you return from that function. Your first task, therefore, is to design an analysis that can detect simple errors like deadlock, which will occur if the programmer tries to lock the same lock twice. For this assignment, you need only consider one thread running at a time–believe it or not, double-locking errors due to a single thread that forgets to unlock a lock have been found in the Linux kernel, causing the system to hang.

You study the problem first in the context of the WHILE language. You model locks with two new kinds of statements:

- lock(x) locks the variable x
- unlock(x) unlocks the variable x

You decide you will base your analysis on a tuple lattice, with one element of the tuple for each lock variable in the program.

Question 1.1 (12 points).

> Design a lattice for a single variable. Your lattice should be able to represent both locked and unlocked states. Define the lattice by giving (a) the set of lattice elements and (b) the ordering relation between them, (c) the top element and (d) the bottom element.

Question 1.2 (4 points).

> What is the initial analysis information before the first statement of each function? Justify your choice (more than one answer may be correct, so long as it is justified).

Question 1.3 (8 points).

> Define the flow functions for your analysis, using the notation given in class. Naturally, you will need to include flow functions for the new `lock(x)` and `unlock(x)` statements.

Question 1.4 (8 points).

Design a visitor-based analysis that will examine the results of lock analysis (above) to identify deadlock errors, where a program locks a variable that is already locked. Specifically, describe (a) the While AST element you will visit to find the error, (b) the analysis information indicating a definite deadlock error, and (c) whether your condition is based on the analysis information immediately before or after the AST element. Finally, (d) explain what you would change about the condition to find a *possible* deadlock error (e.g. in cases where the analysis is too imprecise to tell if there is definitely an error).

There is a corresponding double-unlocking error that could be found, but finding it is not required for this assignment.

Question 1.5 (8 points).

Simulate your analysis on the following program, using the worklist algorithm. Use the notation from the lecture 7, slide 26 ("Example of Worklist"), so you have 4 columns: the first describing to which statement you are applying the flow function (with 0 at the beginning to show the dataflow values at the entry of the CFG), the second column showing the statements on the worklist, and the last two columns showing lock lattice values for each variable (x and y—the variable b is not relevant since it is not a lock).

[lock(x)]$_1$;
if [$b > 0$]$_2$
  then [lock(y)]$_3$
  else [skip]$_4$;
[lock(x)]$_5$;
if [$b > 0$]$_6$
  then [unlock(y)]$_7$
  else [skip]$_8$;
[unlock(x)]$_9$;

2 Bounds Analysis (60 points)

Your next task is to develop an analysis that can bound a variable from above or below by a constant. If the constant is the size of an array, for example, this analysis can prove that an index into that array is within bounds (and thereby show the absence of array bounds errors). The analysis is parameterized by the constant, which we will call C.

You decide you will base your analysis on a tuple lattice, with one element of the tuple for each variable in the program.

Question 2.1 (16 points).

> Design a lattice for a single variable. Your lattice should be able to represent cases where the variable is less than, greater than, or equal to the constant, as well as disjunctions of these, such as less than or equal to. Define the lattice by giving (a) the set of lattice elements and (b) the ordering relation between them, (c) the top element and (d) the bottom element.

Question 2.2 (4 points).

> What is the initial analysis information before the first statement of each function? Justify your choice (more than one answer may be correct, so long as it is justified).

In order to achieve adequately precise results, your analysis will need to take into consideration the conditions of if and while statements. For example, if an if statement tests that a variable is less than constant C, your analysis should recognize that in the "then" branch of the if, the variable must be less than C, whereas in the "else" branch of the if, the variable must be greater than or equal to C. This is particularly important for loops, where the loop index may be used to access an array, and where the boolean condition for the loop may be whether the loop index is within bounds.

Zero analysis can be used to illustrate this principle. For boolean expressions, we can have the analysis return two results, one for when the expression evaluates to true, and one for when it evaluates to false. For example, we might define the following cases for zero analysis, in addition to those discussed in lecture:

$$f^T_{\text{ZA}}(\sigma, [[x]_n = [0]_m]_k) = [x \mapsto Z]\ \sigma$$
$$f^F_{\text{ZA}}(\sigma, [[x]_n = [0]_m]_k) = [x \mapsto NZ]\ \sigma$$

The two cases above apply only to a conditional expression that has the equality operator with a variable x on the left and the number 0 on the right (to be symmetric, we could define equivalent cases where the order is reversed). Other conditional expressions fall under the generic case that does not modify σ.

When applying the analysis to code, we calculate two dataflow results for each conditional to which separate f_{ZA}^T and f_{ZA}^F results apply. If the conditional is directly used as the condition of an if statement or while statement, we use the T results in the then clause of an if or the body of a while, while we use the F results in the else clause of an if or the exit of a whille. If the conditional is nested inside some other conditional expression–e.g. a conjunction or disjunction–we merge the T and F results together (e.g. to produce $[x \mapsto Z]\sigma$ in the example above). If an if or while statement has a conditional that does not fit a pair of f_{ZA}^T and f_{ZA}^F rules, we simply use the f_{ZA} rule on both control flow branches as usual.

Question 2.3 (12 points).

> Define the flow functions for your analysis, using the notation given in class. Your analysis should define special cases of the flow functions for comparing a variable to the constant C, using either the $<$, $=$, or $>$ operators, with separate analysis information on the true and false branches of the statement.

Question 2.4 (8 points).

> Using the notation from the lectures, define an abstraction function mapping a concrete program environment η to analysis state σ. Note that you are mapping from a map of variables to values, to a tuple lattice element (i.e. a map from variables to single variable lattice elements).

Question 2.5 (10 points).

Draw the AST for the program below. On top of this, draw arrows for the control flow graph as shown in class. Use a different notation (i.e. either color or dashed) for the control flow graph arrows, as opposed to the arrows showing the structure of the AST.

```
$[i := 0]_1$;
$while\ [i < C]_2\ do$
  $[d := i]_3$;
  $[i := i + 1]_4$;
$[x := i]_5$
```

Question 2.6 (10 points).

Simulate your analysis on the program above for the constant C used in the program, using the worklist algorithm. Use the notation from the lecture 7, slide 26 ("Example of Worklist"), so you have 4 columns: the first describing to which statement you are applying the flow function (with 0 at the beginning to show the dataflow values at the entry of the CFG), the second column showing the statements on the worklist, and the last two columns showing bounds lattice values for each variable (d and i).

For full credit, your analysis should be precise enough to discover that $d < C$ inside the body of the loop, so that if d were used as an index into an array of size C within the loop, the array index would not be greater than the array size. Note that we could run the same analysis (perhaps with some additional flow function special cases) to determine that $d \geq 0$, which would fully verify the safety of this index.

5 points extra credit if your flow functions are precise enough to determine that $x = C$ at the end of the program. Note that this will require additional careful thought about the flow functions both for $<$ and for $+$. To complete the extra credit, you will need to make the assumption that $C \geq 0$ (otherwise x might not be equal to C).